Protégé Frames: Graph Widget Tutorial

The Graph Widget is an alternative to Forms for creating and populating instances of classes in Protégé. It is also particularly useful for visualizing networks of instances and relationships between instances. This tutorial assumes a basic working knowledge of Protégé. For more information about how to use Protégé, see our User's Guide or Getting Started

---

Getting Started

Step 1: Download GraphWidgetExample project

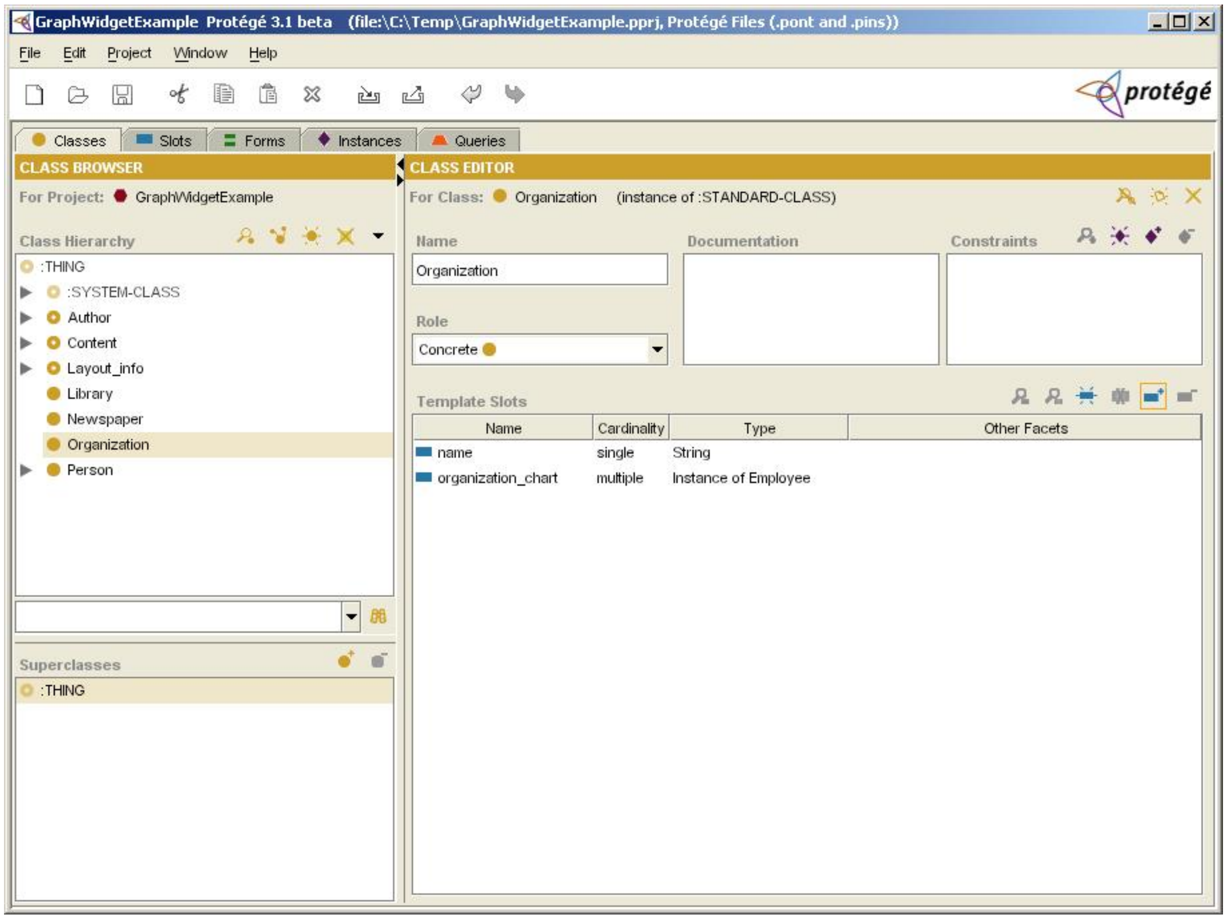
For those of you who are familiar with Protégé, you'll recognize this project as a slightly modified version of the Newspaper example. We'll step through the creation of an organization chart for a group of Newspaper employees.

Download Project

Step 2: Configure Organization class

- Open Protégé and load the GraphWidgetExample project from step 1.
- Choose the `Organization` class in the Class Hierarchy pane on the Classes tab.
- Create a new slot called `organization_chart` and attach it to the `Organization` class. The slot should be of type Instance, cardinality multiple, and have the `Employee` class as the allowed class type. We chose `Employee` as the allowed class type because an organization chart typically displays relationships between employees working within an organization.

Screenshot of Protégé after completion of step 2:


GraphWidgetExample Protégé 3.1 beta (file:\C:\Temp\GraphWidgetExample.pprj, Protégé Files (.pont and .pins))

For Project: GraphWidgetExample

Class Hierarchy
- :THING
  - :SYSTEM-CLASS
  - Author
  - Content
  - Layout_info
  - Library
  - Newspaper
  - Organization
  - Person

Superclasses
- :THING

For Class: Organization (instance of :STANDARD-CLASS)

Name: Organization

Role: Concrete

Documentation

Constraints

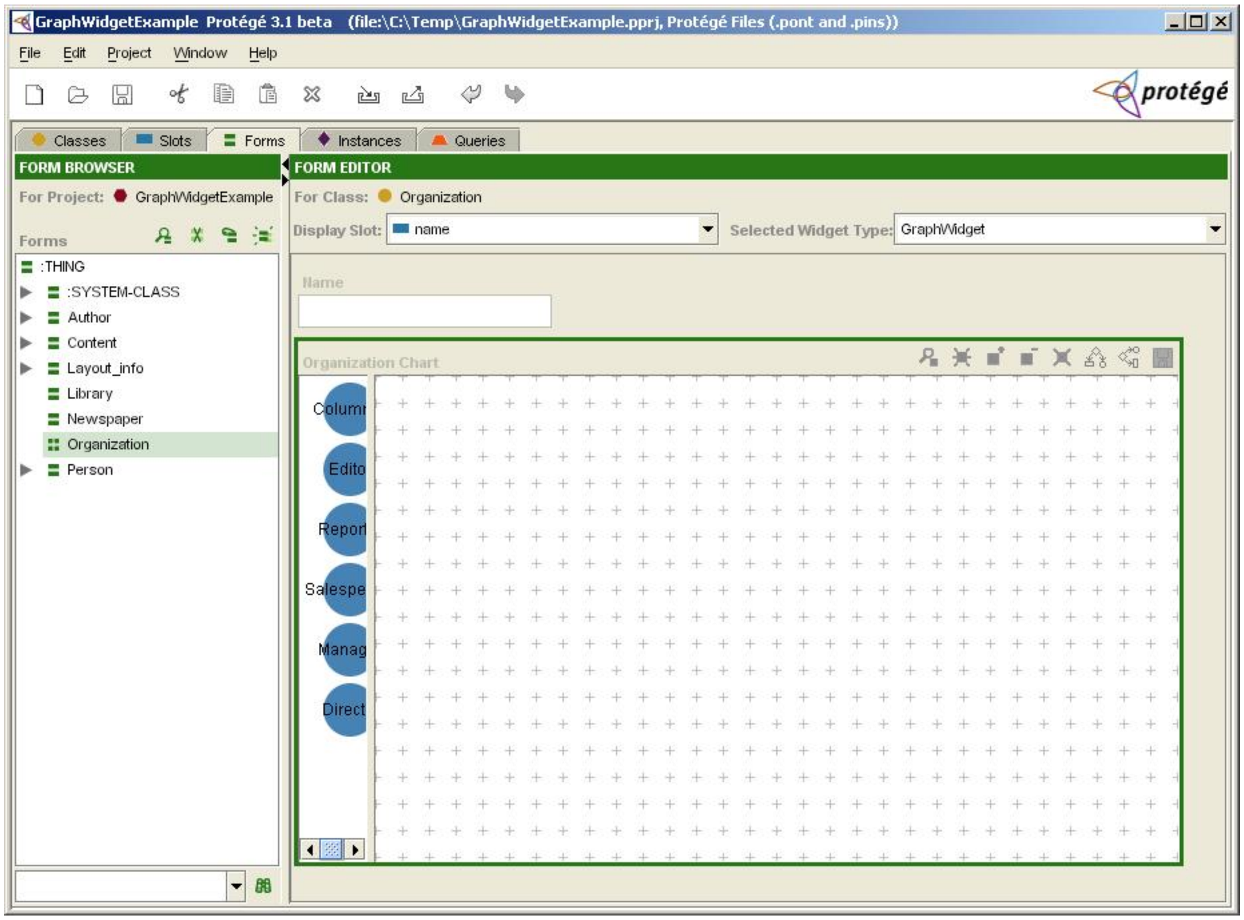
Template Slots

Name	Cardinality	Type	Other Facets
name	single	String	
organization_chart	multiple	Instance of Employee	

Step 3: Configure `organization_chart` slot to use Graph Widget

- Navigate to the Forms tab and choose `Organization` from the Forms list.
- Select the `organization_chart` slot and choose GraphWidget from the Selected Widget Type combo box.

Screenshot of Protégé after completion of step 3:


GraphWidgetExample Protégé 3.1 beta (file:\C:\Temp\GraphWidgetExample.pprj, Protégé Files (.pont and .pins))

For Project: GraphWidgetExample

For Class: Organization

Display Slot: name

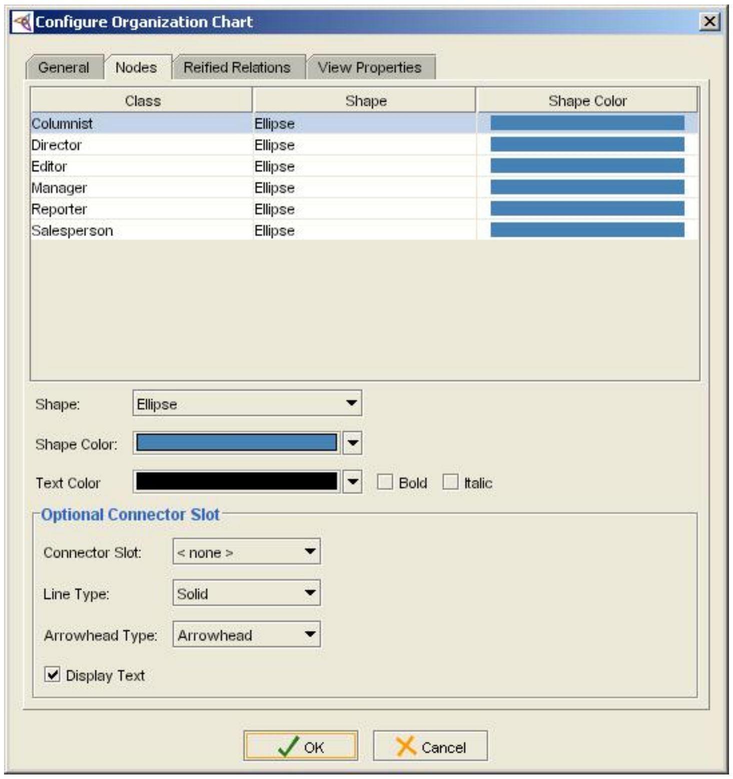
Selected Widget Type: GraphWidget

Step 4: Configure node appearance

Nodes are the objects that appear in the palette on the left-hand side of the Graph Widget. Each node represents a concrete allowed class for the `organization_chart` slot (or whatever slot is configured to use the Graph Widget). By default, the Graph Widget assigns all nodes the same shape and color. To change the node shape and color or the text properties of the node's labels:

- Double-click on the `organization_chart` slot to bring up the Graph Widget configuration dialog.
- Select the Nodes tab.

Screenshot of the Graph Widget configuration dialog:



Class	Shape	Shape Color
Columnist	Ellipse	🟦
Director	Ellipse	🟦
Editor	Ellipse	🟦
Manager	Ellipse	🟦
Reporter	Ellipse	🟦
Salesperson	Ellipse	🟦

Shape: Ellipse

Shape Color: 🟦

Text Color ⬛ ☐ Bold ☐ Italic

Optional Connector Slot

Connector Slot: < none >

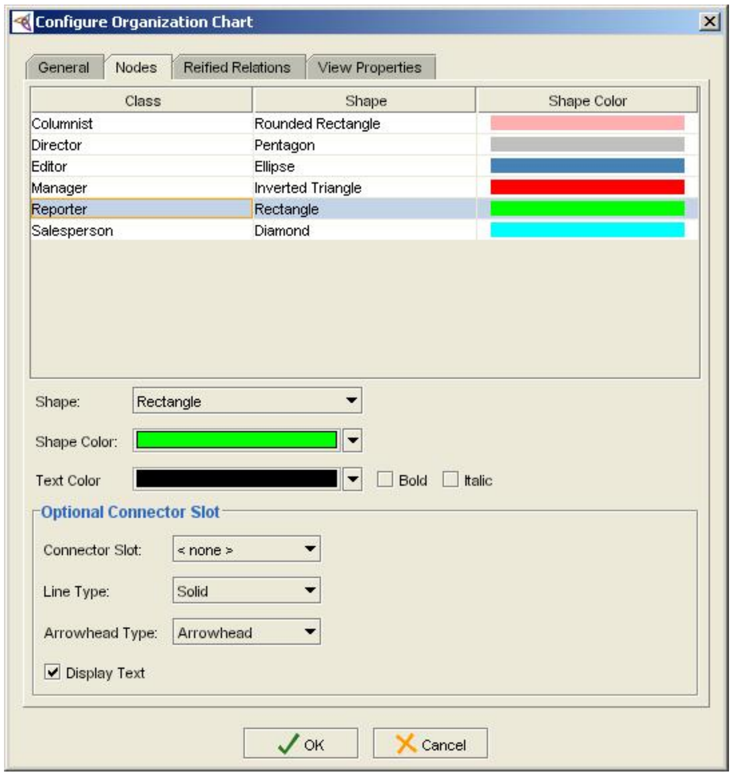
Line Type: Solid

Arrowhead Type: Arrowhead

☑ Display Text

The table in the top portion of the Nodes tab contains a list of all the concrete allowed classes for the `organization_chart` slot. Note that the `Employee` class does not appear here because it's an abstract class. Rather, all the the concrete subclasses of `Employee` are displayed. Try selecting various nodes from the table and changing their shape and color using the Shape and Shape Color combo boxes. You can also change the text properties of the node's labels using the Text Color combo box and the Bold and Italic check boxes.

You can ignore the Optional Connector Slot area as we will deal with this in a later step. To follow is a example of what the configuration dialog might look like after configuring node appearance:



Class	Shape	Shape Color
Columnist	Rounded Rectangle	🟥
Director	Pentagon	⬜
Editor	Ellipse	🟦
Manager	Inverted Triangle	🟥
Reporter	Rectangle	🟩
Salesperson	Diamond	🟦

Shape: Rectangle

Shape Color: 🟩

Text Color ⬛ ☐ Bold ☐ Italic

Optional Connector Slot

Connector Slot: < none >

Line Type: Solid

Arrowhead Type: Arrowhead

☑ Display Text

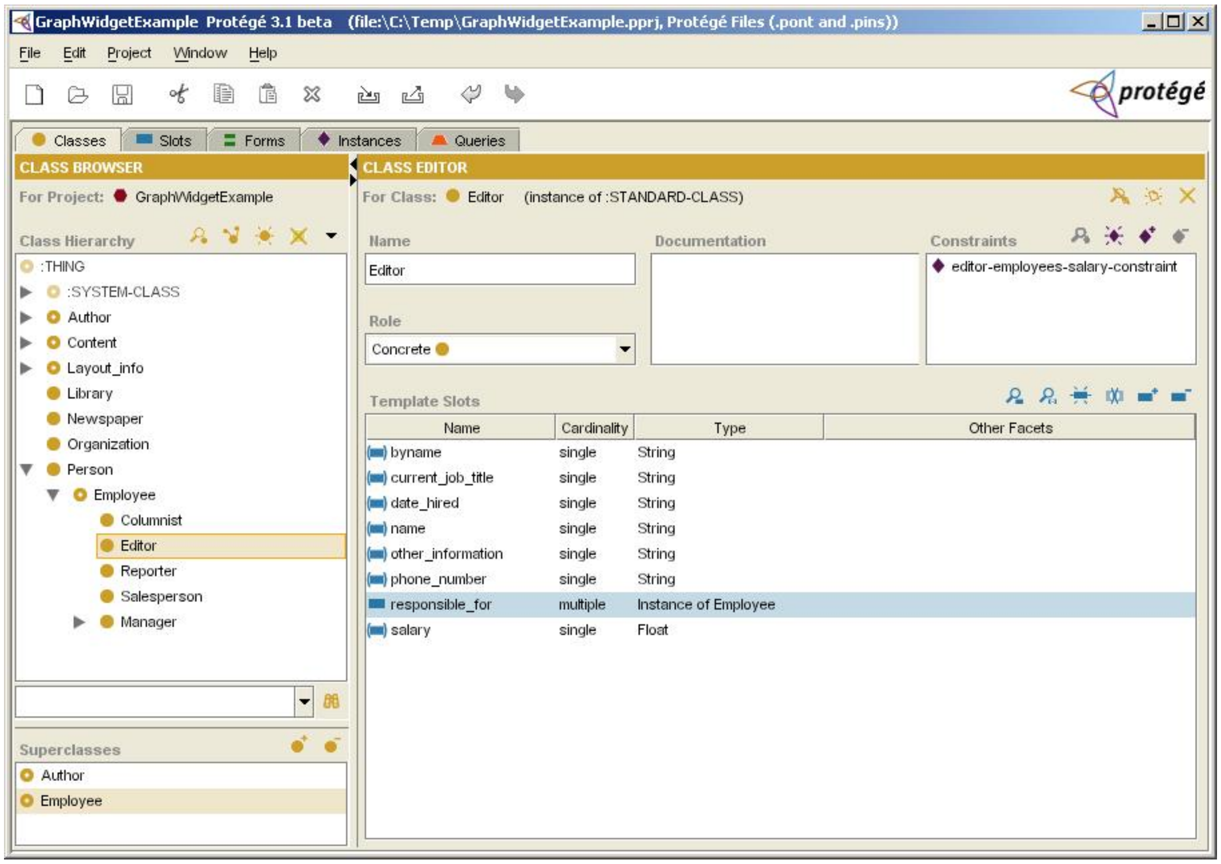
Click OK to save your changes. *(Note that it is not a necessary step in this tutorial to select the same shapes and colors displayed in the screenshot above).*

Step 5: Configure a simple connector

There are two types of connectors in the Graph Widget. First we will look at the simple connector type, which is a connector between two nodes that doesn't have an underlying instance. The allowed class type for the `organization_slot` is `Employee`. Let's assume that we are interested in focusing on the `Editor` subclass of `Employee` and seeing a graphical representation of the relationship between editors and other employees in the newspaper organization. There are two steps to complete for configuring a simple connector:

a. Choose the `Editor` class in the Class Heirarchy pane on the Classes tab and create and attach a slot called `responsible_for`. The slot should be of type Instance, cardinality multiple, and have the `Employee` class as the allowed class type.

   Screenshot of Protégé after completion of step 5a:


GraphWidgetExample Protégé 3.1 beta (file:\C:\Temp\GraphWidgetExample.pprj, Protégé Files (.pont and .pins))

File Edit Project Window Help

Classes | Slots | Forms | Instances | Queries

CLASS BROWSER

For Project: GraphWidgetExample

Class Hierarchy

- :THING
  - :SYSTEM-CLASS
  - Author
  - Content
  - Layout_info
  - Library
  - Newspaper
  - Organization
  - Person
    - Employee
      - Columnist
      - Editor
      - Reporter
      - Salesperson
      - Manager

Superclasses

- Author
- Employee

CLASS EDITOR

For Class: Editor (instance of :STANDARD-CLASS)

Name: Editor

Documentation

Constraints: editor-employees-salary-constraint

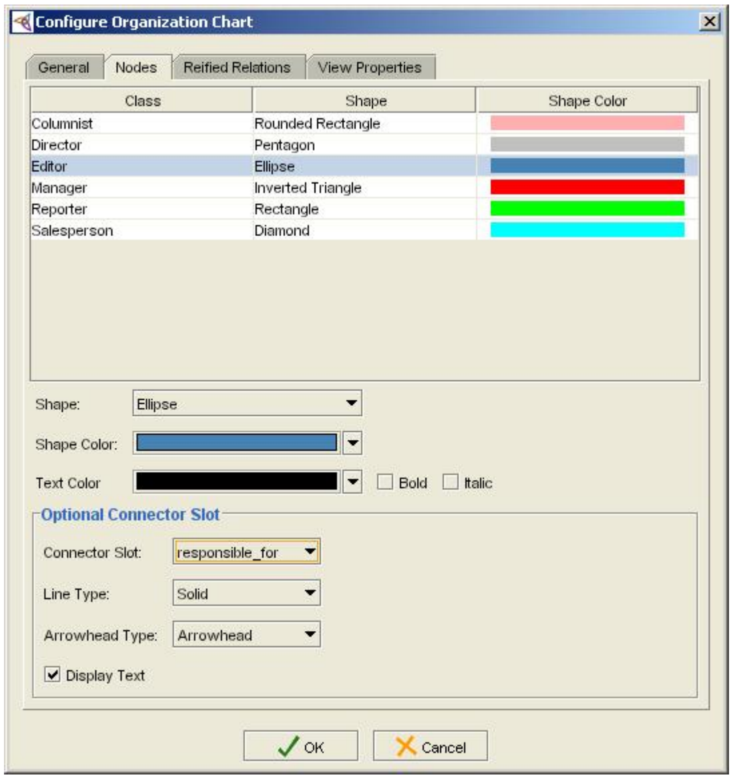
Role: Concrete

Template Slots

Name	Cardinality	Type	Other Facets
byname	single	String	
current_job_title	single	String	
date_hired	single	String	
name	single	String	
other_information	single	String	
phone_number	single	String	
responsible_for	multiple	Instance of Employee	
salary	single	Float	

b. On the Forms tab, bring up the widget configuration dialog again and select the `Editor` class on the Nodes tab. Choose `responsible_for` from the Connector Slot combo box. By making this selection, we are indicating to the Graph Widget that we want to be able to draw connectors between `Editor` nodes and other nodes of type `Employee`. In other words, for each allowed class displayed in the table on the upper portion of the Nodes tab, the Connector Slot combo box will display a list of type Instance, cardinality multiple slots for that class. You can choose one of these slots from the combo box to indicate what other class types you can draw connectors to. Click OK to save your changes.

Screenshot of widget configuration dialog after completion of step 5b:



Class	Shape	Shape Color
Columnist	Rounded Rectangle	🟥
Director	Pentagon	⬜
Editor	Ellipse	🟦
Manager	Inverted Triangle	🟥
Reporter	Rectangle	🟩
Salesperson	Diamond	🟦

Shape: Ellipse

Shape Color: 🟦

Text Color ⬛ ☐ Bold ☐ Italic

Optional Connector Slot

Connector Slot: responsible_for

Line Type: Solid

Arrowhead Type: Arrowhead

☑ Display Text

Step 6: Populate an instance of the Graph Widget using nodes and simple connectors

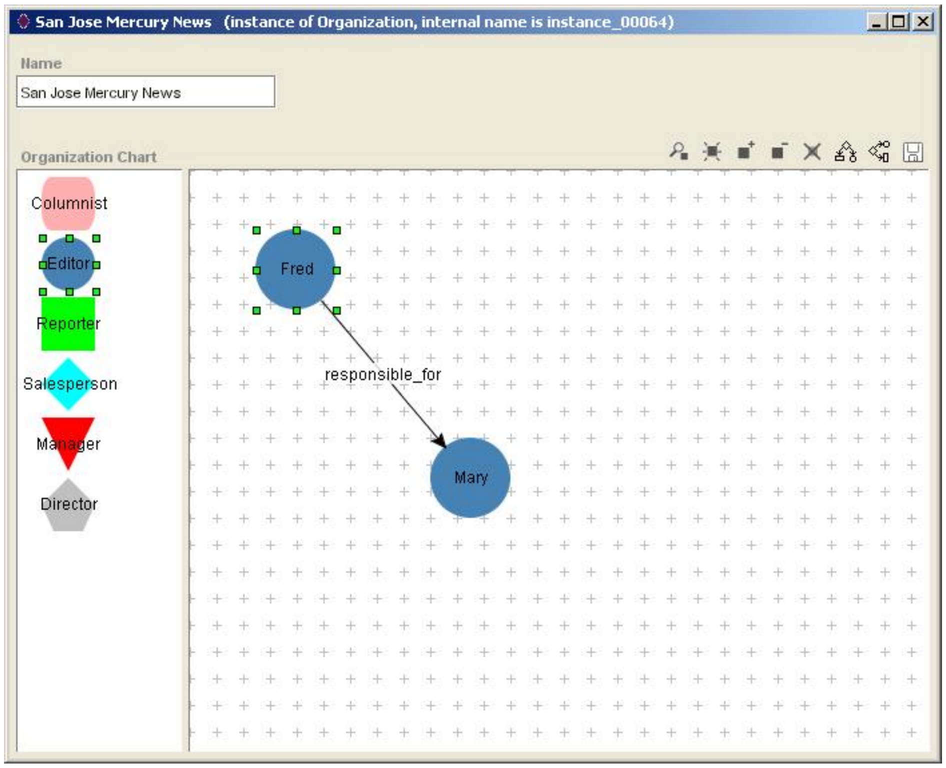
In this step, we'll look at an instance of the `Organization` class and learn how to populate the Graph Widget with nodes and simple connectors.

Navigate to the Instances tab and choose the `Organization` class in the Class Hierarchy pane. Select San Jose Mercury News in the Instance Browser. To create an instance of the `Editor` class, select the `Editor` node in the palette and drag and drop it onto the view. Repeat this step again so that you have two nodes in the Graph Widget view. To draw a simple connector between the two `Editor` nodes, mouse over one of the nodes in an area other than the node label. When the cursor changes from an arrow to a hand, single-click and drag your mouse over to the other node. When you release your mouse over the other node, a connector appears between the two nodes, showing the `responsible_for` relationship.

Tips:

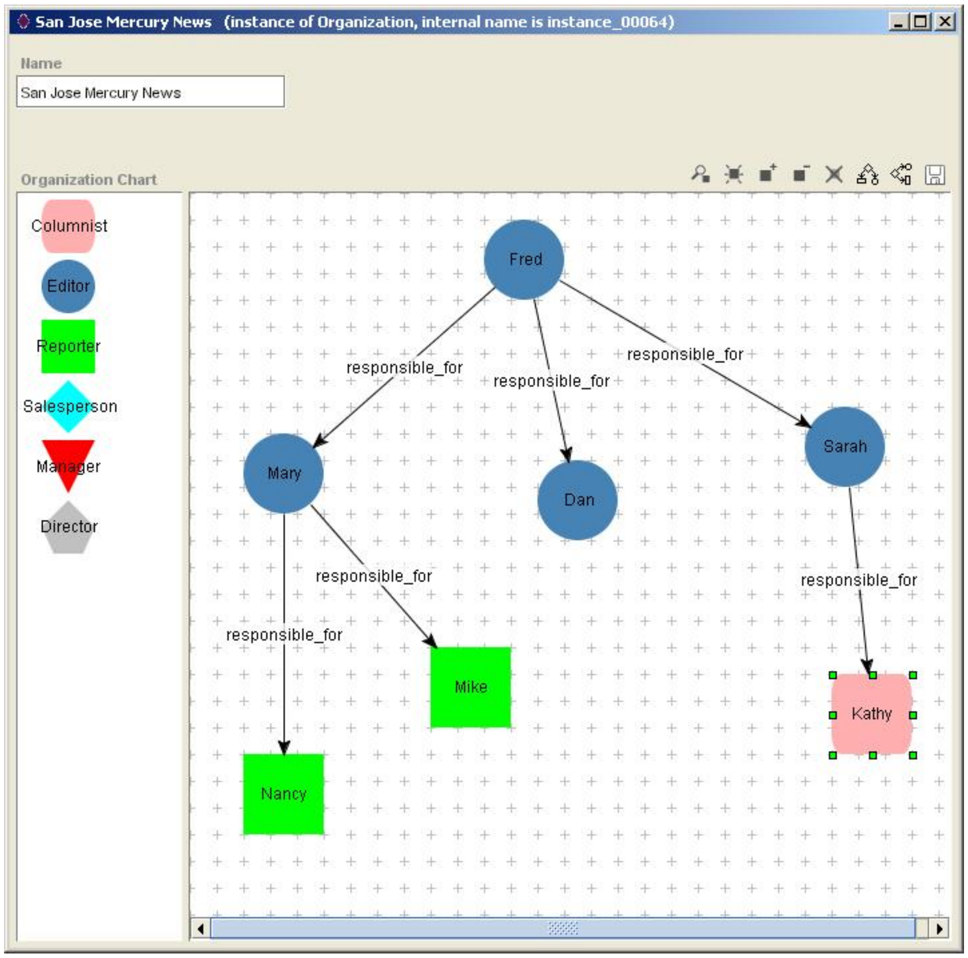
- Click on a node label and drag to reposition nodes in the view.
- When a node is selected, grab any of the eight resize handles to resize the node.
- Since the Editor class has a browser key (the `name` slot), you can double-click on the node labels and use in-place text editing to change the node's text.
- Double-click on nodes (anywhere except the node label) to bring up Instance Forms.
- Use the ´C´ button on the Graph Widget toolbar to create nodes in the view as an alternative to dragging and dropping from the palette.
- Select multiple nodes and click the ´V´ button to bring up multiple Instance Forms.
- Right click on connectors to insert extra points via the Insert Point right-mouse menu item.
- Right click on connectors to remove segments via the Remove Segment right-mouse menu item.

Screenshot of San Jose Mercury News Instance Form after completion of step 6:



Continue adding nodes of various types to the Graph Widget. Notice also that the widget will only allow you to draw valid simple connectors between nodes. Since we only specified one connector slot for the `Editor` class, we can only draw simple connectors *from* `Editor` nodes *to* other nodes, but not the other way around. For example, try drawing a simple connector from a `Columnist` or `Reporter` node to another node. The Graph Widget won't allow such a connection.

Additional screenshot of SJ Mercury News Instance Form after adding more nodes and connectors:



Tips:

- Use the "Perform Automatic Layout" buttons to automatically layout nodes and connectors in the Graph Widget.
- Use the "Save Graph as Image" button to save the contents of the Graph Widget to an image file.

Step 7: Configure a reified relation

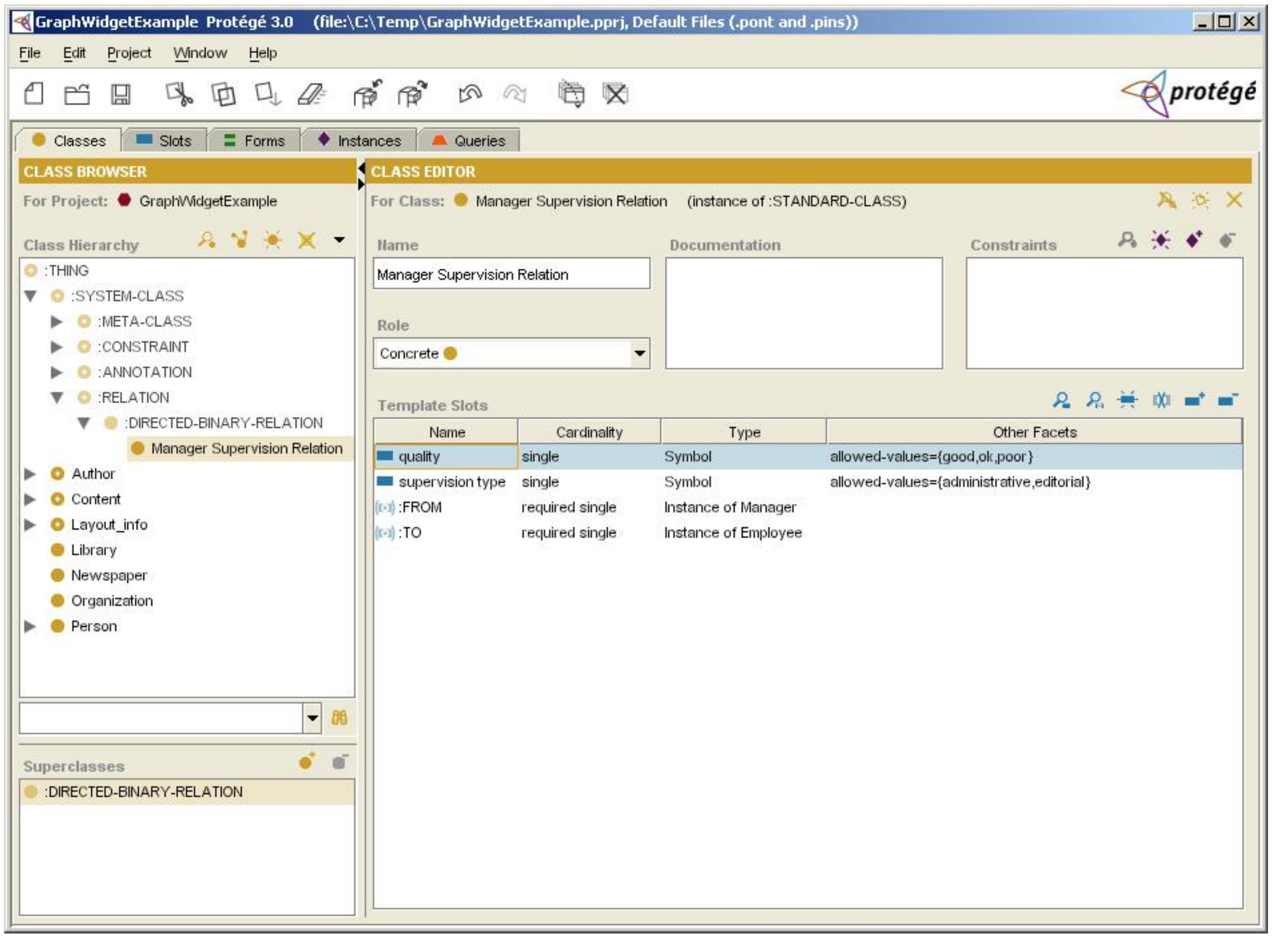
In this step we will work with the other connector type provided by the Graph Widget called a reified relation. A reified relation is a connector between nodes that has an underlying instance. You use this type of connector when you need to store additional information about the relationship between two nodes. There are three steps to configure a reified relation:

a. Subclass the :DIRECTED-BINARY-RELATION system class.

   - Choose the :DIRECTED-BINARY-RELATION system class in the Class Hierarchy pane on the Classes tab.
   - Create a subclass called `Manager Supervision Relation`.
   - Change the allowed class type of the :FROM slot to `Manager`.
   - Change the allowed class type of the :TO slot to `Employee`.

   What we have just defined is a class that represents a reified relation between `Manager` and `Employee` nodes. Note that you can add additional slots to this class to keep track of other information about the relationship. The screenshot below has examples of what sort of "other information" you might want to add. Note that it's not a necessary step in this tutorial to add additonal slots to the `Manager Supervision Relation` class.

   Screenshot of Protégé after completion of step 7a:



GraphWidgetExample Protégé 3.0 (file:\C:\Temp\GraphWidgetExample.pprj, Default Files (.pont and .pins))

For Project: GraphWidgetExample

Class Hierarchy

- :THING
  - :SYSTEM-CLASS
    - :META-CLASS
    - :CONSTRAINT
    - :ANNOTATION
    - :RELATION
      - :DIRECTED-BINARY-RELATION
        - Manager Supervision Relation
  - Author
  - Content
  - Layout_info
  - Library
  - Newspaper
  - Organization
  - Person

Superclasses: :DIRECTED-BINARY-RELATION

For Class: Manager Supervision Relation (instance of :STANDARD-CLASS)

Name: Manager Supervision Relation

Documentation

Constraints

Role: Concrete

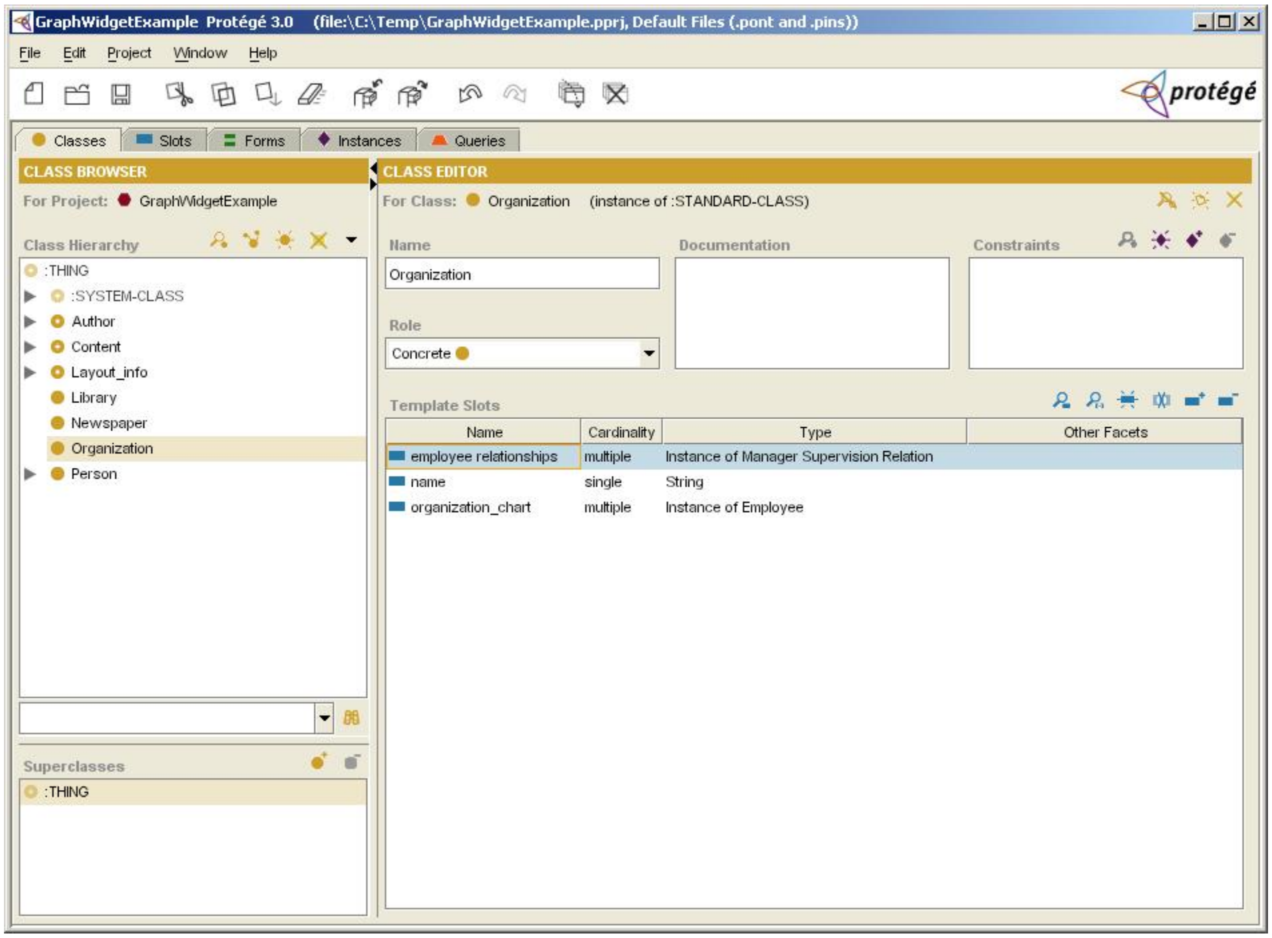
Template Slots

Name	Cardinality	Type	Other Facets
quality	single	Symbol	allowed-values={good,ok,poor}
supervision type	single	Symbol	allowed-values={administrative,editorial}
:FROM	required single	Instance of Manager	
:TO	required single	Instance of Employee	

b. Create a slot to hold reified relations.

If you want to use reified relations for a particular class, you need to add a slot to the class that designates what the valid relations are for that class. For our example, add a slot to the `Organization` class called `employee_relationships`. The slot should be of type Instance, cardinality multiple, and have the `Manager Supervision Relation` class as the allowed class type.

Screenshot of Protégé after completion of step 7b:


GraphWidgetExample Protégé 3.0 (file:\C:\Temp\GraphWidgetExample.pprj, Default Files (.pont and .pins))
File Edit Project Window Help
Classes Slots Forms Instances Queries
CLASS BROWSER
CLASS EDITOR
Class Hierarchy
:THING
:SYSTEM-CLASS
Author
Content
Layout_info
Library
Newspaper
Organization
Person
Superclasses
:THING

For Project: GraphWidgetExample

For Class: Organization (instance of :STANDARD-CLASS)

Name: Organization

Role: Concrete

Documentation

Constraints

Template Slots

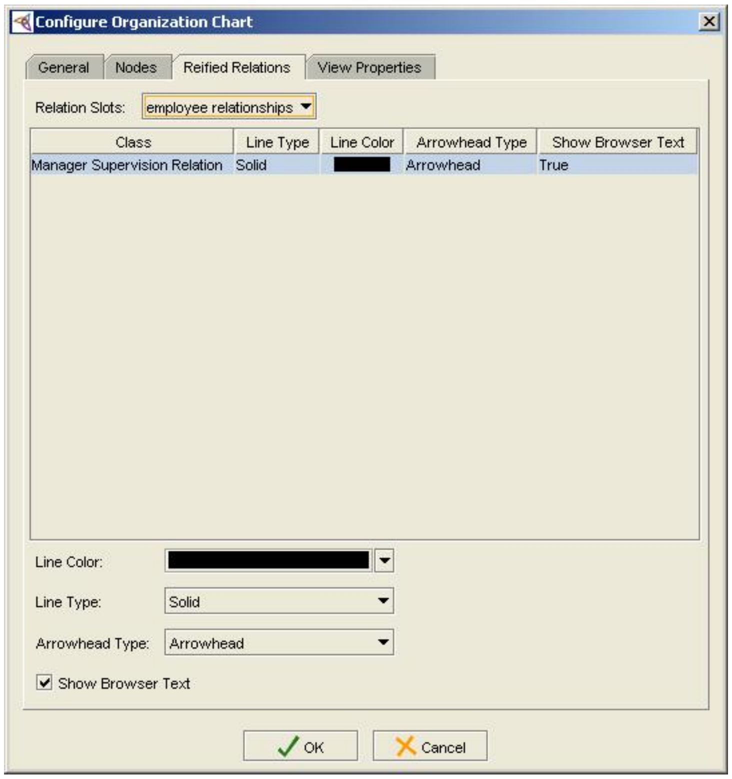
Name	Cardinality	Type	Other Facets
employee relationships	multiple	Instance of Manager Supervision Relation	
name	single	String	
organization_chart	multiple	Instance of Employee	

c. Tell the Graph Widget which of the slots attached to your class holds reified relations.

   We need to tell the Graph Widget which slot attached to the `Organization` class has been created to hold reified relations. This is done in the widget configuration dialog:

   - Return to the Forms tab and bring up the widget configuration dialog.
   - Click on the Reified Relations tab.
   - Choose `employee_relationships` from the Relation Slots combo box.

     Screenshot of widget configuration dialog after completion of step 7c:



Relation Slots: employee relationships

Class	Line Type	Line Color	Arrowhead Type	Show Browser Text
Manager Supervision Relation	Solid	■	Arrowhead	True

Line Color: ■

Line Type: Solid

Arrowhead Type: Arrowhead

☑ Show Browser Text

Tips:

- Use the Line Color, Line Type, and Arrowhead Type combo boxes to change the appearance of the reified relation connectors.
- Uncheck the Show Browser Text check box if you don't want your reified relation connectors to have labels on them.

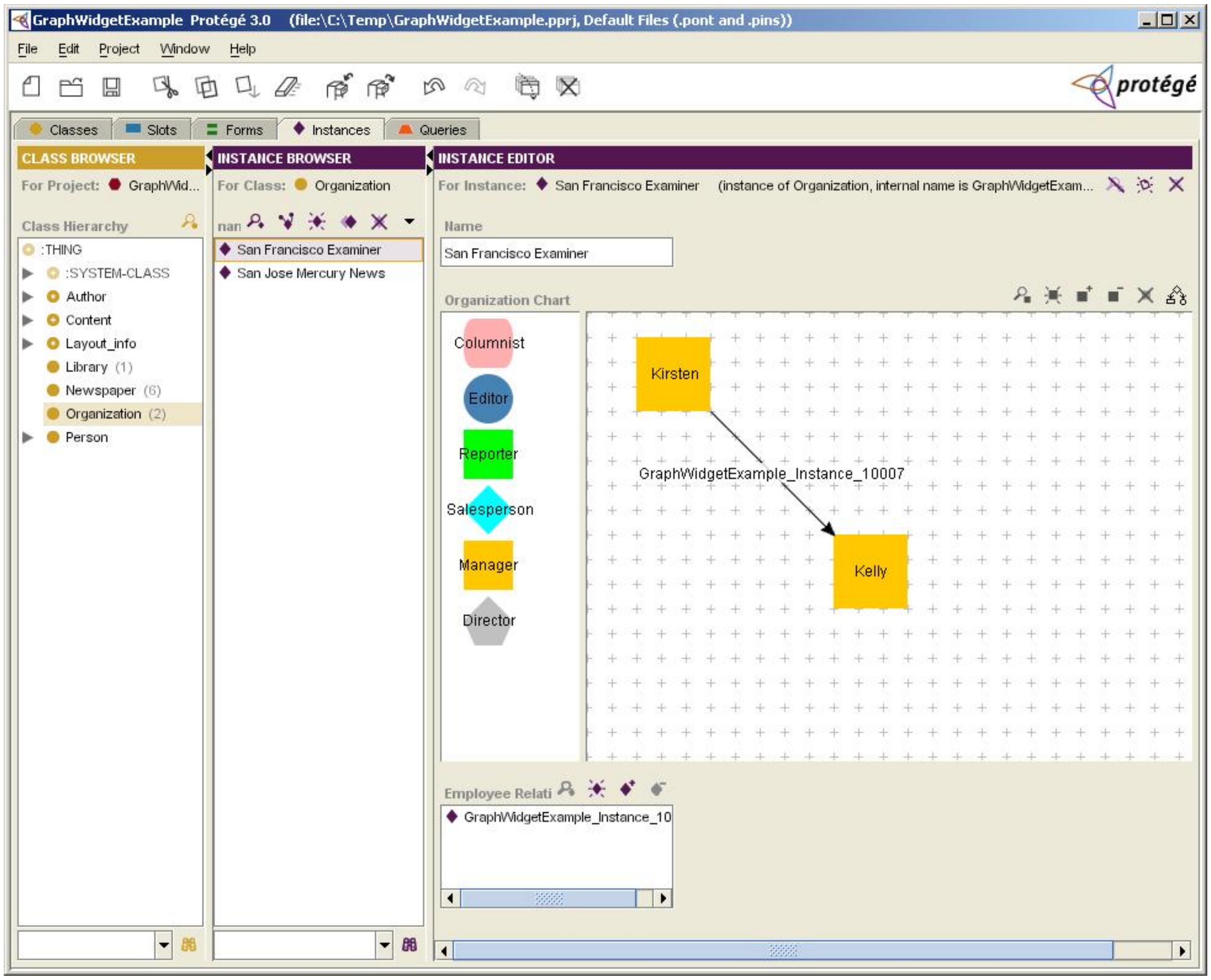
- Click OK to save your changes.

Step 8: Populate an instance of the Graph Widget using nodes and reified relations

In this step, we'll look at a new instance of the `Organization` class and learn how to populate the Graph Widget with nodes and reified relations.

Choose the `Organization` class in the Class Hierarchy pane on the Instances tab. Create a new instance of the `Organization` class called `San Francisco Examiner`. Using the Graph Widget, create two instances of the `Manager` class by dragging and dropping the `Manager` node from the palette onto the view. To draw a reified relation between the two nodes, mouse over one of the nodes in an area other than the node label. When the cursor changes from an arrow to a hand, single-click and drag your mouse over to the other node. When you release your mouse, a reified relation appears between the two nodes, showing a managerial relationship.

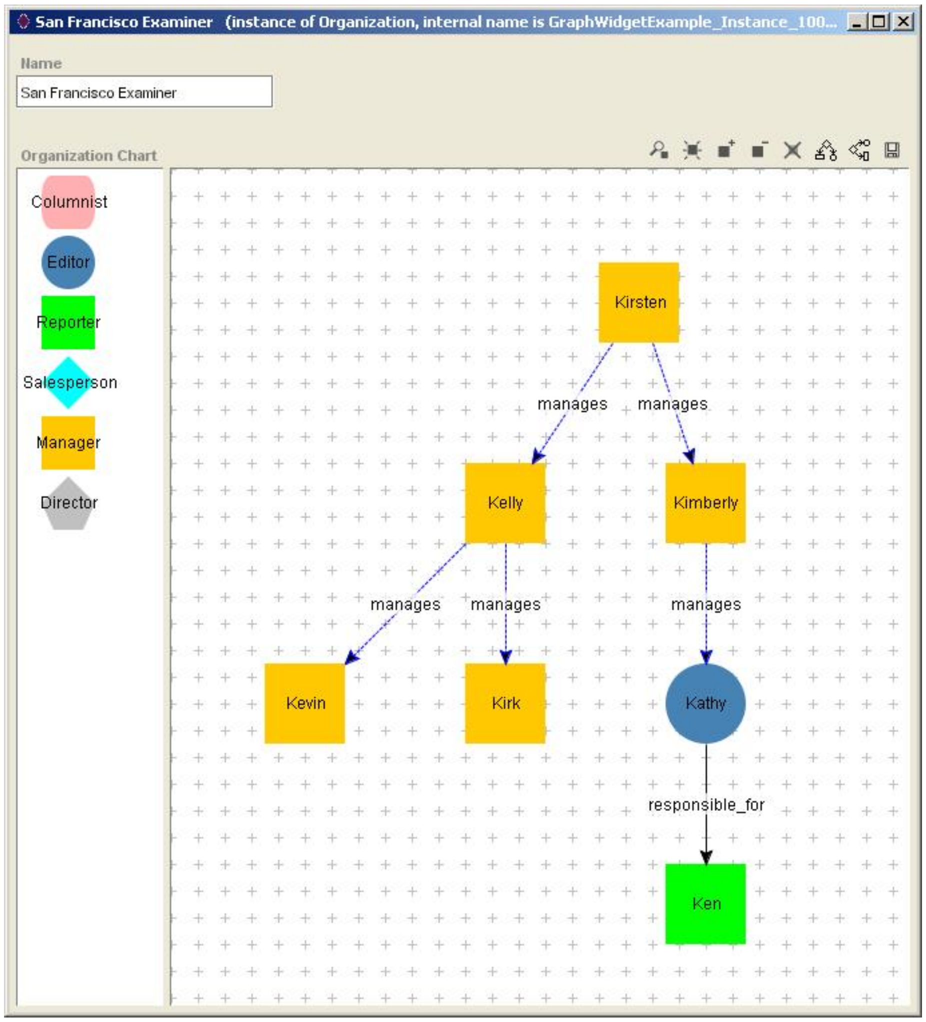
Screenshot of Protégé after completion of step 8:



Tips:

- Double click on the reified relation connector to bring up the Instance Form for the `Manager Supervision Relation` class.
- Designate a display slot for the `Manager Supervision Relation` class to enable in-place editing of the reified relation connector's label. (One possibility would be to attach the `name` slot).
- On the Forms tab, set the Selected Widget Type for the `employee_relationships` slot to <none> to expand the amount of display for the Graph Widget.

Continue adding nodes and experimenting with reified relations. To follow is a screenshot of the San Francisco Examiner Instance Form after adding more nodes, reified relations, a simple connector, and completing the tips specified above:



Congratulations! You have now completed the Graph Widget tutorial. If you have further questions, please post them to the protege-user mailing list.

Converting from the Diagram Widget to the Graph Widget

The Graph Widget is the new and improved version of the Diagram Widget. The Diagram Widget existed in earlier versions of Protégé (versions 1.7 and earlier). If you have an old Protégé project with diagram instances, you may want to read the following notes (courtesy of Samson Tu) on how to convert existing instances of the Diagram Widget to the Graph Widget.

Assumptions:

- You've installed Protégé version 1.8, build 1032 or later.
- You're interested in saving existing diagrams.
- You're using a file system back-end.

1st Scenario:

You have a project that includes the diagram ontology, that has a number of diagram instances, and you do not need to use the existing location information of nodes and connectors.

1. Make the `:DIRECTED-BINARY-RELATION` class the parent class of your classes that inherit from the diagram widget `Connector` class. Your existing subclasses of the `Connector` class will then inherit the `:FROM` and `:TO` slots from the `:DIRECTED-BINARY-RELATION` class.
2. In your existing subclasses of the Network class, add a multi-valued slot (you can choose your own name), whose allowed classes include the allowed classes of the existing `connectors` slot. This slot will be used to hold arc information in the new Graph Widget.
3. Save the project.
4. Open the pins file in a text editor.
5. Globally replace `first_object` with `:FROM` and `second_object` with `:TO` (assuming that you are not using the `first_object` or `second_object` slots for anything else).
6. Rename the slot `connectors` in your instances that hold the diagram to the new slot you've defined in step 2.
7. Save the pins file.
8. Reload the project.
9. Change the metaclasses of your old `Connector` and `Network` classes to `:STANDARD_CLASS` (or any metaclass your want).
10. Drop `Connector` and `Network` classes as parents of your classes.
11. Delete instances of `ObjectLocation`, `Rectangle`, and `Point`.
12. In the Forms tab, configure appropriate classes and slots in your project (that used to be shown as diagrams) to use the new Graph Widget.

Do not "uninclude" the diagram project from your own project until you have converted all projects that include the project where you specify the classes that make use of the diagram ontology.

2nd Scenario:

You have a project (Project A) that includes another project (Project B) where diagrams are configured and you do not need to use the existing location information of nodes and connectors.

1. Convert Project B where you specify the classes that make use of the diagram ontology (see 1st scenario above).
2. In a text editor, open the pins file for Project A. Globally replace `first_object` with `:FROM` and `second_object` with `:TO` (assuming that you are not using the `first_object` or `second_object` slots for anything else).
3. Rename the slot `connectors` in your instances that hold the diagram to the new slot name you've defined.
4. Save the pins file.
5. You may have to reconfigure the classes of Project A in the Forms tab so that the Graph Widget gets used appropriately. Alternatively, you can "import" the pont and pins files to rebuild the pprj file. When a new pprj file is built, Protégé will copy the widget configuration information from included projects.

Known Bugs

- Graph Widget & the Mac: The Graph Widget only works on Mac OS X Panther v10.3 with an upgraded JDK (to version 1.4.2).
- There are some cases where the positioning information is lost for nodes and connectors the first time you click away from a new instance of the graph widget. The workaround for this bug is to click away from new instances of the graph widget and come back to them before beginning to populate them with new nodes and connectors.

Known Feature Requests

- The ability to specify icons instead of shapes to represent nodes.
- An additional arrowhead type of circle
- The ability to designate arrowhead type for source arrowhead (can only specify destination arrowhead now)
- The ability to control the color of nodes based on the value of a specified slot
- Copy and paste for nodes within an instance of the graph widget
- Copy and paste for nodes between different instances of the graph widget

Troubleshooting new nodes

- If your node has an empty text label, there will still be a spot in the center of the node that you can mouse over where the cursor will change from a hand to an arrow so you can single click on the node to select it. This is also the spot that you single-click on and drag if you want to move the node around the view.
- If your node has an empty text label and you want to add some text back in, mouse-over the center of the node until the cursor changes from a hand to an arrow. Double-click on this spot to initiate in-place text editing. You can

also still add text back in using the node's instance form. See above for instructions on launching instance forms.

- If the node's text label is so large that it obscures the node's shape, this can affect the ability to draw connectors between nodes. The reason for this is that you must mouse over the node in an area other than the text label in order for the cursor to change from an arrow to a hand (the signal that you can single-click and drag to start drawing a connector). The workaround for this is to click on the node label to select the node, grab any of the resize handles and resize the node until the node's bounding rectangle is large enough to accomodate the label and allow extra space for the node's shape. You can either leave the node larger or after you are done drawing all connectors that initiate from the node, you can resize it down to it's orginal size.

Information about the graphics library used by the Graph Widget

Rather than reinvent the wheel (write our own graphics library), we used a third party graphics library called JGo from Northwoods Software. JGo has no runtime fees but if you'd like to extend the GraphWidget and you need access to the JGo library, contact them to purchase a license.